

Complexity of the path avoiding forbidden pairs problem revisited

Jakub Kováč

Department of Computer Science, Comenius University, Mlynská Dolina,
842 48 Bratislava, Slovakia

Abstract

Let $G = (V, E)$ be a directed acyclic graph with two distinguished vertices s, t , and let F be a set of *forbidden pairs* of vertices. We say that a path in G is *safe*, if it contains at most one vertex from each pair $\{u, v\} \in F$. Given G and F , the *path avoiding forbidden pairs* (PAFP) problem is to find a safe s - t path in G .

We systematically study the complexity of different special cases of the PAFP problem defined by the mutual positions of forbidden pairs. Fix one topological ordering $<$ of vertices; we say that pairs $\{u, v\}$ and $\{x, y\}$ are *disjoint*, if $u < v < x < y$, *nested*, if $u < x < y < v$, and *halving*, if $u < x < v < y$.

The PAFP problem is known to be NP-hard in general or if no two pairs are disjoint; we prove that it remains NP-hard even when no two forbidden pairs are nested. On the other hand, if no two pairs are halving, the problem is known to be solvable in cubic time. We simplify and improve this result by showing an $O(M(n))$ time algorithm, where $M(n)$ is the time to multiply two $n \times n$ boolean matrices.

Keywords: path, forbidden pairs, NP-hard, dynamic programming

1. Introduction

Let $G = (V, E)$ be a directed graph with two distinguished vertices $s, t \in V$ and let $F \subseteq V \times V$ be a set of *forbidden pairs* of vertices. We say that a path π is *safe*, if it does not contain any forbidden pair, i.e., π contains at most one vertex from each pair $\{u, v\} \in F$. Given G and F , the *path avoiding forbidden pairs* problem (henceforth PAFP) is to find a safe s - t path in G . In this paper, we study the complexity of different special cases of the problem on directed acyclic graphs.

1.1. Motivation

The PAFP problem was first studied by Krause et al. [1] and Srimani and Sinha [2] motivated by designing test cases for automatic software testing and validation. We can represent a program as a directed graph where vertices represent segments of code and edges represent the flow of control from one code segment into another. The goal is to cover this graph with s - t paths corresponding to different test cases. However, not all paths correspond to executable sequences in the program. Therefore Krause et al. [1] introduced forbidden pairs which identify the mutually exclusive code segments and formulated the PAFP problem. Unfortunately, as shown by Gabow et al. [3], the problem is NP-hard even for directed acyclic graphs.

A different motivation came from bioinformatics and the problem of peptide sequencing via tandem mass spectrometry. Peptides are polymers which can be thought of as strings over a 20 character alphabet of amino acids and the sequencing problem is to determine the amino acid sequence of a given peptide. To this end, many copies of the peptide are fragmented and the mass of the fragments is measured (very precisely) by mass spectrometer. The result of the experiment is a mass spectrum where each peak corresponds to mass of some prefix or some suffix of the amino acid sequence, or is a noise. The spectrum is then compared against a database of known fragment weights.

Email address: kuko@ksp.sk (Jakub Kováč)

Chen et al. [4] suggested the following formulation of the peptide sequencing problem: Let us create a *spectrum graph* with two vertices p_i and s_i for each peak w_i with weights $w(p_i) = w_i - 1$ and $w(s_i) = W - w_i + 1$, where W is the weight of the whole peptide. We add an edge from x to y if the difference between weights $w(y) - w(x)$ equals the total mass of some known sequence of amino acids. Thus, paths in this graph correspond to amino acid sequences. Paths going through p_i correspond to w_i being a weight of some prefix and similarly, paths going through s_i correspond to w_i being a weight of some suffix. (Paths going through neither p_i nor s_i correspond to w_i being a noise.) However, w_i cannot be a prefix weight *and* a suffix weight at the same time, so $\{p_i, s_i\}$ will form a forbidden pair for each i . This is a very special case of the PAFP problem in directed acyclic graphs where all the forbidden pairs are nested and Chen et al. [4] showed that it is polynomially solvable.

The PAFP problem on directed acyclic graphs also arose in a completely different application in bioinformatics – gene finding using RT-PCR tests [5]. In this application, we have a so called *splicing graph* where vertices represent non-overlapping segments of the DNA sequence, length of a vertex is the number of nucleotides in this segment, and edge (u, v) indicates that segment v immediately follows segment u in some gene transcript. Thus, paths in this splicing graph correspond to putative genes. The problem is to identify the true genes with a help of information from RT-PCR experiments.

Without going into biology details, let us define a (simplified) result of an RT-PCR experiment as a triple $t = (u, v, \ell)$, where $u, v \in V$ are two vertices and ℓ is the length of a product. Let π be a path going through u and v in the splicing graph; if the length of the u – v subpath is equal to ℓ , we say that π *explains* test t , otherwise, it is *inconsistent* with test t . We can define a score of a path π with respect to a set of tests T as a sum of the scores of all of its vertices and edges, plus a bonus B for each explained test from T , and minus a penalty P for each inconsistent test. The *gene finding with RT-PCR tests* problem is to find an s – t path with the highest score in the given splicing graph G with a set of RT-PCR tests T .

Note that if we set all lengths to an unattainable value, say -1 , and we set a high (infinite) penalty P for inconsistent tests, we basically get the PAFP problem. Thus, the PAFP problem is at the core of gene finding with RT-PCR tests and the latter problem inherits all NP-hardness results for the PAFP problem. On the positive side, we have shown in our previous work [5] that some polynomial solutions for special cases of the PAFP problem can be extended to pseudo-polynomial algorithms for the gene finding problem.

1.2. Previous results

As shown by Gabow et al. [3], the PAFP problem is NP-hard in general, but several special cases are polynomially solvable. Yannone [6] studied the PAFP problem under *skew symmetry* conditions where for each two forbidden pairs $\{u, u'\}, \{v, v'\} \in F$, if there is an edge from u to v , there is also an edge from v' to u' . He proved that under such conditions, the problem is polynomially equivalent to finding an augmenting path with respect to a given matching and thus polynomially solvable.

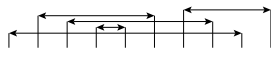
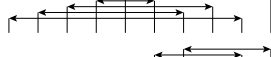
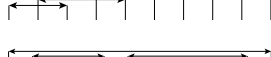

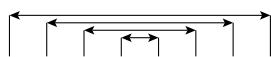
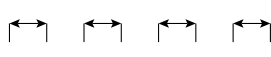

For directed acyclic graphs, we have already mentioned that the nested case is solvable in polynomial time [4]; Kolman and Pangrác [7] were able to devise a polynomial algorithm if the set of forbidden pairs has a well-parenthesized or a halving structure (see Preliminaries).

Recently, approximability and parameterized complexity of the PAFP problem have been studied: We add 1 to the objective function to disallow a zero cost solutions – otherwise the problem is trivially inapproximable. Hajiaghayi et al. [8] showed that even then there is a constant $c > 0$ such that minimizing $1 +$ the number of forbidden pairs on an s – t path is not $c \cdot n$ -approximable. Bodlaender et al. [9] studied the PAFP problem on undirected graphs. When parameterized by the vertex cover of $G = (V, E)$, the problem is W[1]-hard (the proof also carries over to directed acyclic graphs). On the other hand, when parameterized by the vertex cover of $H = (V, F)$ (where edges are forbidden pairs), the problem is fixed parameter tractable (FPT), but has no polynomial kernel unless $\text{NP} \subseteq \text{coNP/poly}$. The problem is also FPT when parameterized by the treewidth of $G \cup H$.

1.3. Contributions and road map

In this paper, we systematically study different special cases of the PAFP problem on directed acyclic graphs. In the next section, we introduce the different special cases based on mutual positions of forbidden pairs. In Section 3, we prove that the PAFP problem is NP-hard even if the set of forbidden pairs has ordered structure and in Sections 4 and 5, we improve upon the results of Chen et al. [4] and Kolman and Pangrác [7] for the nested, halving, and well-parenthesized forbidden pairs.

Table 1: Complexity of the PAFP problem for its different special cases; n and m denote the number of vertices and edges of G , respectively; $O(n^\omega)$ is the complexity of boolean matrix multiplication, $\omega < 2.3727$ [10, 11].

PROBLEM	ALLOWED FORBIDDEN PAIRS			COMPLEXITY	EXAMPLE
	<i>disjoint</i>	<i>nested</i>	<i>halving</i>		
<i>general problem</i>	✓	✓	✓	NP-hard [3]	
<i>overlapping structure</i>	×	✓	✓	NP-hard [7]	
<i>ordered</i>	✓	×	✓	NP-hard [new]	
<i>well-parenthesized</i>	✓	✓	×	$O(n^3)$ [7], $O(n^\omega)$ [new]	
<i>halving</i>	×	×	✓	$O(n^5)$ [7], $O(n^{\omega+1})$ [new]	
<i>nested</i>	×	✓	×	$O(nm)$ [4], $O(n^\omega)$ [new]	
<i>disjoint</i>	✓	×	×	$O(n + m)$ [trivial]	

2. Preliminaries

Let $G = (V, E)$ be a directed acyclic graph and let F be the set of forbidden pairs. As already noticed by Yinnone [6] and Kolman and Pangráč [7], we may assume that every vertex except for s and t belongs to exactly one forbidden pair, i.e., $\bigcup F = V - \{s, t\}$. This is simply because if vertex v does not belong to any forbidden pair, we can remove it and replace all 2-edge paths u, v, w by a direct edge (u, w) . On the other hand, if v belongs to $k > 1$ forbidden pairs, we can replace it by a directed path of length k and move the ends of forbidden pairs to different vertices on this path.

To define special cases of interest, we fix one topological ordering of vertices. We say that vertex u is *before* or *precedes* v , $u < v$, if u precedes v in this linear order. Let us denote the forbidden pairs $\{f_i, f'_i\}$ for $i = 1, \dots, k$, where $f_i < f'_i$ and $f_1 < f_2 < \dots < f_k$, i.e., we order them by position of the left member of the pair.

We recognize three possible types of mutual position of pairs $\{u, v\}$ and $\{x, y\}$ (without loss of generality, let $u < v$, $x < y$, and $u < x$): *disjoint* ($u, v < x, y$; see Fig. 1(a)), *nested* ($u < x, y < v$; see Fig. 1(b)), and *halving* ($u < x < v < y$; see Fig. 1(c)). All the special cases are obtained by restricting the set of forbidden pairs F to only certain types of mutual positions (see Table 1). This gives us $2^3 = 8$ cases, from which these 6 classes are non-trivial and interesting:

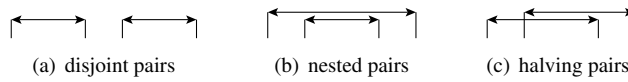


Figure 1: Different mutual positions of two forbidden pairs.

1. *general case* – there are no constraints on the positions of pairs;
2. *overlapping structure*¹ – every two forbidden pairs overlap (they may be nested or halving, but not disjoint); as a consequence, $f_1 < f_2 < \dots < f_k < f'_{\sigma(1)} < f'_{\sigma(2)} < \dots < f'_{\sigma(k)}$ for some permutation σ ;
3. *ordered* – there may be disjoint and halving pairs, but no two forbidden pairs are nested; as a consequence $f_1 < f_2 < \dots < f_k$ and $f'_1 < f'_2 < \dots < f'_k$;
4. *well-parenthesized* – there may be disjoint and nested pairs, but no two pairs are halving; this case deserves its name since if we write $($ and $)$ for the i -th pair, we get a well-parenthesized sequence;
5. *halving* – every two pairs halve each other; $f_1 < f_2 < \dots < f_k < f'_1 < f'_2 < \dots < f'_k$;
6. *nested* – there are only nested pairs i.e., the vertices in forbidden pairs are ordered $f_1 < f_2 < \dots < f_k < f'_k < \dots < f'_2 < f'_1$; this is a special case of the well-parenthesized case.

¹note that this special case is referred to as *halving structure* by Kolman and Pangráč [7]; we reserve the term “halving” for sets where every two pairs halve each other

The previous work and our own results are summarized in Table 1.

For completeness and as a warm-up, we include our own proof of NP-hardness of the PAFP problem in the general and overlapping case. This proof is also simpler than the one given by Kolman and Pangrác [7].

Theorem 1. *The PAFP problem is NP-hard, even when the set of forbidden pairs has overlapping structure.*

Proof. By reduction from 3-SAT: Let $\phi = \bigwedge_{1 \leq i \leq n} \phi_i$ be a formula over m variables x_1, \dots, x_m , with n clauses $\phi_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$, where each literal $\ell_{i,j}$ is either x_k or $\neg x_k$. We will construct graph G and a set of forbidden pairs F such that there is an s - t path avoiding pairs in F if and only if ϕ is satisfiable.

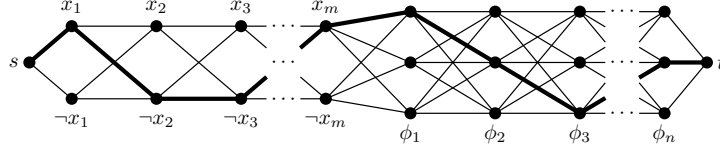


Figure 2: Input for the PAFP problem for the formula $\phi_1 \wedge \phi_2 \wedge \dots \wedge \phi_n$. All edges are directed from left to right.

G consists of two parts: The first part contains a vertex for each variable x_k and its negation $\neg x_k$ (see Fig. 2). A path traversing this first part corresponds to a truth assignment of variables where the visited vertices are true. The second part contains a vertex for each literal $\ell_{i,j}$ (see Fig. 2). Forbidden pairs connecting every literal from the first part to every occurrence of its negation in the second part of G will ensure that we can only go through “true” vertices. Thus an s - t path avoiding F exists if and only if every clause is satisfied. Since every forbidden pair starts in the first part and ends in the second part, all pairs overlap. \square

3. Ordered forbidden pairs

In this section, we turn to a seemingly more restricted version of the PAFP problem, allowing only disjoint and halving forbidden pairs. This special case has not been studied before.

Theorem 2. *The PAFP problem is NP-hard, even when the set of forbidden pairs is ordered.*

Proof. We will prove the claim by reduction from 3-SAT. Let ϕ be a logical formula over m variables x_1, \dots, x_m , which is a conjunction of n clauses $\phi_1 \wedge \dots \wedge \phi_n$, where $\phi_i = (\ell_{i,1} \vee \ell_{i,2} \vee \ell_{i,3})$ and each literal $\ell_{i,j}$ is either x_k or $\neg x_k$. We will construct graph G with a linear order $<$ on its vertices and an ordered set of forbidden pairs F such that there is an s - t path avoiding pairs in F if and only if ϕ is satisfiable.

Graph G consists of several blocks B and B_ℓ of $2m$ vertices shown in Fig. 3(a), (b). The blocks are connected together as outlined in Fig. 3(c). Any left-to-right path through the block B naturally corresponds to a truth assignment of the variables and, since B_ℓ has an isolated vertex $\neg \ell$, a path through block B_ℓ corresponds to an assignment where ℓ is true. A clause gadget consists of three such blocks, each corresponding to one literal. Any s - t path must pass through one of the three blocks, and thus choose an assignment that satisfies the clause. The forbidden pairs in F will enforce that the assignment of the variables is the same in all blocks. This is done by adding a forbidden pair between all literals ℓ' in the B_ℓ -blocks with their counterparts $\neg \ell'$ in the previous and the following B -block.

The order of literals in a B -block is $\neg x_1 < x_1 < \neg x_2 < \dots < x_m$, while the order in a B_ℓ -block is $x_1 < \neg x_1 < x_2 < \dots < \neg x_m$. Let $v_1^i < v_2^i < v_3^i < \dots$ be the order of vertices in graph G^i . A *zipping* operation takes graphs G^1, G^2, G^3 and produces a new graph $G^1 \cup G^2 \cup G^3$ with vertices ordered $v_1^1 < v_2^1 < v_3^1 < v_2^2 < v_2^3 < v_3^2 < \dots$. The clause gadgets are produced by zipping the three blocks corresponding to their literals. If we do not allow multiple forbidden pairs starting or ending in the same vertex, we can substitute vertices in G for short paths as in Fig. 3(d). It is easy to check that under such linear order, no two pairs in F are nested. \square

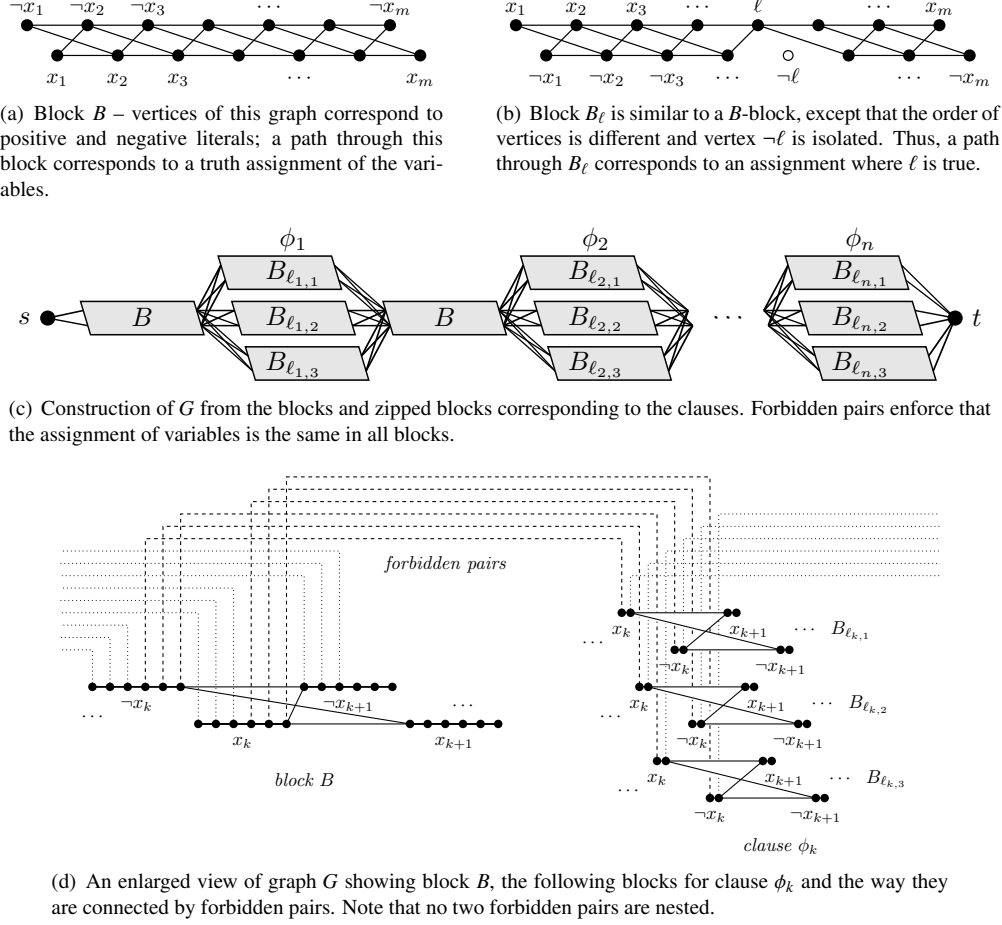


Figure 3: Construction of the graph G for a 3-SAT formula ϕ . All edges are directed from left to right.

4. Well-parenthesized forbidden pairs

The first polynomial algorithm for the PAFP problem with well-parenthesized forbidden pairs was given by Kolman and Pangrác [7]. Their algorithm uses three rules for reducing the input graph:

1. *contraction of a vertex* – if v does not appear in any forbidden pair, remove it and add a direct edge (u, w) for every pair of edges $(u, v), (v, w)$;
2. *removal of an edge* – if edge $e \in E \cap F$ joins two vertices that make up a forbidden pair, remove e from E ;
3. *removal of a forbidden pair* – if $(u, v) \in F$ is a forbidden pair, but there is no path from u to v , remove (u, v) from F .

These three rules are alternately applied to the input graph until we end up with vertices s and t only – either joined by an edge or disconnected – which is a trivial problem.

A simple implementation of this approach gives an $O(n^2m)$ algorithm. Using fast matrix multiplication, the time complexity can be reduced to $O(n^{\omega+1}) \approx O(n^{3.373})$ and using a dynamic data structure for “finding paths and deleting edges in directed acyclic graphs” by Italiano [12], it can be reduced still to $O(n^3)$.

Here we describe our own $O(n^3)$ algorithm, its advantages being simplicity, extensibility, and improvability: The algorithm does not use any advanced data structures. It can be easily extended to solve problems such as

- find an s – t path passing the minimum number of forbidden pairs or

- given a graph where all edges have scores and there are bonuses or penalties for some (well-parenthesized) pairs of vertices, find an s - t path with maximum score (this problem was considered by Kováč et al. [5]).

It seems unlikely that these problems can be solved using the former approach (because of rule 2). Furthermore, our algorithm can be improved using the Valiant's technique and fast matrix multiplication algorithms [13, 14] or the Four-Russians technique [15]. Note that the reduction to matrix multiplication is not only of theoretical interest, since there are fast and practical hardware-based solutions for multiplying two matrices [16, 17].

Theorem 3. *The PAFP problem with well-parenthesized forbidden pairs can be solved in $O(n^3)$ time.*

Proof. We modify the input graph so that no two forbidden pairs start or end in the same vertex. Let $P[u, v]$ be true if a safe u - v path exists and let $J[u, v]$ be true if there is a forbidden pair $(q, v) \in F$, $u < q < v$, and there is a safe u - v path such that the first edge jumps over q .

The values of P and J can be found by dynamic programming: It is easy to compute $J[u, v]$ (if we already know $P[w, v]$ for all $u < w \leq v$) by inspecting the neighbours of u and conversely, we can also compute $P[u, v]$ efficiently using the table J : If no forbidden pair ends in v or vertex u is “inside” the forbidden pair $(q, v) \in F$, we just search the neighbours of v for a vertex that could be penultimate on the u - v path. Otherwise, let $(q, v) \in F$ be a forbidden pair such that $u < q < v$. Suppose that a safe u - v path exists and let w be the last vertex on this path before q . Then $P[u, w]$ and $J[w, v]$ are both true. Conversely, if $P[u, w]$ and $J[w, v]$ are true for some $w < q$, by concatenating the corresponding paths, we get a safe u - v path: The path obviously avoids all forbidden pairs before or after q (from the definition of $P[u, w]$ and $J[w, v]$) and there are no forbidden pairs halving (q, v) .

Thus, $P[s, t]$ can be computed in cubic time using the following two recurrences:

$$J[u, v] = \begin{cases} \bigvee_{(u, w) \in E, q < w} P[w, v] & \text{if } u < q \text{ and } (q, v) \in F \text{ is a forbidden pair} \\ \text{undefined} & \text{otherwise} \end{cases} \quad (1)$$

$$P[u, v] = \begin{cases} \text{true} & \text{if } u = v \\ \text{false} & \text{if } (u, v) \in F \text{ is a forbidden pair} \\ \bigvee_{u \leq w < v, (w, v) \in E} P[u, w] & \text{if no forbidden pair ends in } v \text{ or } (q, v) \in F \text{ for } q < u \\ \bigvee_{u \leq w < q} (P[u, w] \wedge J[w, v]) & \text{if } (q, v) \text{ is a forbidden pair, } u < q < v \end{cases} \quad (2)$$

$$\quad (3)$$

□

This algorithm can be further improved to $O(n^\omega)$ time by using fast boolean matrix multiplication. The proof is actually simple thanks to the work of Zakov et al. [14] that simplified and generalized the Valiant's technique [13]. They introduce a generic problem called *Inside Vector Multiplication Template* (VMT) which can be solved in subcubic time. A problem is considered an Inside VMT problem if it fulfills the following requirements:

1. The goal of the problem is to compute for every i, j a series of inside properties $\beta_{i,j}^1, \beta_{i,j}^2, \dots, \beta_{i,j}^K$.
2. Let $1 \leq k \leq K$, and let $\mu_{i,j}^k$ be a result of a vector multiplication of the form $\mu_{i,j}^k = \bigoplus_{q \in (i,j)} (\beta_{i,q}^{k'} \otimes \beta_{q,j}^{k''})$, for some $1 \leq k', k'' \leq K$. Assume that the following values are available: $\mu_{i,j}^k$, all values $\beta_{i,j}^{k'}$ for $1 \leq k' \leq K$ and $(i', j') \subsetneq (i, j)$ and all values $\beta_{i,j}^{k'}$ for $1 \leq k' < k$. Then, $\beta_{i,j}^k$ can be computed in $o(n)$ time.
3. In the multiplication variant that is used for computing $\mu_{i,j}^k$, the \oplus operation is associative, and the domain of elements contains a zero element. In addition, there is a matrix multiplication algorithm for this multiplication variant, whose running time $M(n)$ over two $n \times n$ matrices satisfies $M(n) = o(n^3)$.

Theorem 4 (Zakov et al. [14]). *For every Inside VMT problem there is an algorithm whose running time is $o(n^3)$. In particular, let $M(n)$ be the complexity of the matrix multiplication used and suppose that $\beta_{i,j}^k$ can be computed in $\Theta(1)$ time in item 2 of the definition above. Then the time complexity is $\Theta(M(n) \log n)$, if $M(n) = O(n^2 \log^k n)$; and $\Theta(M(n))$, if $M(n) = \Omega(n^{2+\varepsilon})$ for $\varepsilon > 2$ and $4M(n/2) \leq d \cdot M(n)$ for some $d < 1$ and sufficiently large n .*

Corollary 1. *The PAFP problem with well-parenthesized forbidden pairs can be solved in $O(n^\omega)$ time, where $2 < \omega < 2.3727$ is the exponent in the complexity of the boolean matrix multiplication.*

Proof. We formulate our solution from Theorem 3 as an Inside VMT problem. The goal is to compute inside properties $A, J, \alpha, \beta, P, P^{\lceil \bullet \rceil}$, and $P^{\lfloor \bullet \rfloor}$. Properties $J_{u,v}$ and $P_{u,v}$ correspond to the dynamic programming tables from the proof of Theorem 3, other properties are auxiliary. Property A is the adjacency matrix of graph G and it is constant ($A_{u,v} = 1$ if and only if $(u, v) \in E$). Properties α, β are used to store the partial results from cases (2) and (3) in the computation of $P[u, v]$. Finally, the auxiliary properties $P^{\lceil \bullet \rceil}$ and $P^{\lfloor \bullet \rfloor}$ can be computed from P in constant time and are defined as follows:

$$P_{w,v}^{\lceil \bullet \rceil} = P_{w,v} \wedge (q < w) \quad \text{if } (q, v) \in F, \text{ else false} \quad P_{w,v}^{\lfloor \bullet \rfloor} = P_{w,v} \wedge (w < q) \quad \text{if } (q, v) \in F, \text{ else false}$$

Now we can rewrite the computation of $J_{u,v}$ and $P_{u,v}$ using boolean vector multiplication as follows:

$$\bigvee_{(u,w) \in E, q < w} P[w, v] \quad \rightsquigarrow \quad J_{u,v} = \bigoplus_{w \in (u,v)} (A_{u,w} \otimes P_{w,v}^{\lceil \bullet \rceil}) \quad (1')$$

$$\bigvee_{u \leq w \leq v, (w,v) \in E} P[u, w] \quad \rightsquigarrow \quad \alpha_{u,v} = \bigoplus_{w \in (u,v)} (P_{u,w} \otimes A_{w,v}) \quad (2')$$

$$\bigvee_{u \leq w < q} (P[u, w] \wedge J[w, v]) \quad \rightsquigarrow \quad \beta_{u,v} = \bigoplus_{w \in (u,v)} (P_{u,w}^{\lfloor \bullet \rfloor} \otimes J_{w,v}) \quad (3')$$

Property $P_{u,v}$ can be computed from $\alpha_{u,v}$ and $\beta_{u,v}$ in constant time. \square

5. The other cases and concluding remarks

Note that the $O(n^\omega)$ algorithm for well-parenthesized forbidden pairs also improves upon the result by Chen et al. [4] for the nested case. It remains an open problem whether there is a more efficient algorithm for the nested case.

An $O(n^{\omega+1})$ time algorithm for halving forbidden pairs is achieved by a refined version of the algorithm given by Kolman and Pangráč [7]. Recall that in this case, the input graph G consists of two parts: all the forbidden pairs start in the first part, and end in the second part in the same order. Let us denote the vertices in the first part $s < x_1 < \dots < x_n$ and vertices in the second part $y_1 < \dots < y_n < t$, where $\{x_i, y_i\}$ are forbidden pairs. We may assume that all vertices are accessible from s and that t is accessible from every vertex.

If there is a direct edge from s to the second part or if there is an edge from the first part to t , a safe s - t path exists trivially. Otherwise, we reduce the halving case to n instances of the nested case. There will be a safe s - t path in G if and only if there is a safe s - t' path in at least one of the produced instances.

First, remove all the (x_i, y_j) edges, add a new terminal vertex t' , and reverse the direction of all edges in the second part of G . Note that in this new order, $s < x_1 < \dots < x_n < t < y_n < \dots < y_1 < t'$, the forbidden pairs are nested. The k -th instance is obtained by adding edges (x_k, t) and (y_ℓ, t') for each edge (x_k, y_ℓ) , so there is a safe s - t path $s, \dots, x_k, y_\ell, \dots, t$ in the original graph G if and only if there is a safe s - t' path $s, \dots, x_k, t, \dots, y_\ell, t'$ in the new graph.

It remains an open problem whether a more efficient algorithm exists.

Acknowledgements.

The autor would like to thank Broňa Brejová for many constructive comments. The research of Jakub Kováč is supported by APVV grant SK-CN-0007-09 Marie Curie Fellowship IRG-231025 to Dr. Broňa Brejová, Comenius University grant UK/121/2011, and by National Scholarship Programme (SAIA), Slovak Republic. Preliminary version of this work appeared in Kováč et al. [5].

References

- [1] K. Krause, R. Smith, M. Goodwin, Optimal software test planning through automated network analysis, in: Proc. 1973 IEEE Symp. on Computer Software Reliability, 18–22, 1973.
- [2] P. K. Srimani, B. P. Sinha, Impossible pair constrained test path generation in a program, Inf. Sci. 28 (2) (1982) 87–103.
- [3] H. N. Gabow, S. N. Maheswari, L. J. Osterweil, On Two Problems in the Generation of Program Test Paths, IEEE Trans. Software Eng. 2 (3) (1976) 227–231.
- [4] T. Chen, M.-Y. Kao, M. Tepel, J. Rush, G. M. Church, A Dynamic Programming Approach to De Novo Peptide Sequencing via Tandem Mass Spectrometry, J. Comput. Biol. 8 (3) (2001) 325–337.
- [5] J. Kováč, T. Vinař, B. Brejová, Predicting Gene Structures from Multiple RT-PCR Tests, in: S. Salzberg, T. Warnow (Eds.), WABI, vol. 5724 of Lecture Notes in Computer Science, Springer, ISBN 978-3-642-04240-9, 181–193, 2009.
- [6] H. Yinnone, On Paths Avoiding Forbidden Pairs of Vertices in a Graph, Discrete Appl. Math. 74 (1) (1997) 85–92.

- [7] P. Kolman, O. Pangrác, On the complexity of paths avoiding forbidden pairs, *Discrete Appl. Math.* 157 (13) (2009) 2871–2876.
- [8] M. Hajiaghayi, R. Khandekar, G. Kortsarz, J. Mestre, The Checkpoint Problem, in: M. J. Serna, R. Shaltiel, K. Jansen, J. D. P. Rolim (Eds.), *APPROX-RANDOM*, vol. 6302 of *Lecture Notes in Computer Science*, Springer, ISBN 978-3-642-15368-6, 219–231, 2010.
- [9] H. L. Bodlaender, B. M. P. Jansen, S. Kratsch, Kernel Bounds for Path and Cycle Problems, in: *Proc. of the 6th International symposium on Parameterized and Exact Computation (IPEC)*, 2011.
- [10] D. Coppersmith, S. Winograd, Matrix Multiplication via Arithmetic Progressions, *J. Symb. Comput.* 9 (3) (1990) 251–280.
- [11] V. Williams, Breaking the Coppersmith-Winograd barrier, 2011.
- [12] G. F. Italiano, Finding Paths and Deleting Edges in Directed Acyclic Graphs, *Inf. Process. Lett.* 28 (1) (1988) 5–11.
- [13] L. G. Valiant, General Context-Free Recognition in Less than Cubic Time, *J. Comput. Syst. Sci.* 10 (2) (1975) 308–315.
- [14] S. Zakov, D. Tsur, M. Ziv-Ukelson, Reducing the worst case running times of a family of RNA and CFG problems, using Valiant’s approach, *Algorithms Mol. Biol.* 6 (1) (2011) 20.
- [15] V. Arlazarov, E. Dinic, M. Kronrod, I. Faradzev, On economic construction of the transitive closure of a directed graph, in: *Soviet Math. Dokl.*, vol. 11, 1209–1210, 1970.
- [16] S. Ryoo, C. I. Rodrigues, S. S. Baghsorkhi, S. S. Stone, D. B. Kirk, W. mei W. Hwu, Optimization principles and application performance evaluation of a multithreaded GPU using CUDA, in: S. Chatterjee, M. L. Scott (Eds.), *PPOPP*, ACM, ISBN 978-1-59593-795-7, 73–82, 2008.
- [17] V. Volkov, J. Demmel, Benchmarking GPUs to tune dense linear algebra, in: *SC*, IEEE/ACM, ISBN 978-1-4244-2835-9, 31, 2008.